SystemC – Events (08A)

SystemC

Young Won Lim
06/09/2012

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Young Won Lim
06/09/2012

Based on the following original work

[1] Aleksandar Milenkovic, 2002
CPE 626 The SystemC Language – VHDL, Verilog Designer's Guide
http://www.ece.uah.edu/~milenka/ce626-02S/lectures/cpe626-SystemC-L2.ppt

[2] Alexander de Graaf, EEMCS/ME/CAS, 2010
SystemC: an overview ET 4351
ens.ewi.tudelft.nl/Education/courses/et4351/SystemC-2010v1.pdf

[3] Joachim Gerlach, 2001
System-on-Chip Design with Systent of Computer Engineering
http://www2.cs.uni-paderborn.de/cs/ag-hardt/Forschung/Data/SystemC-Tutorial.pdf

[4] Martino Ruggiero, 2008
SystemC
polimage.polito.it/~lavagno/codes/SystemC_Lezione.pdf

[5] Deepak Kumar Tal, 1998-2012
SystemC Tutorial
http://www.asic-world.com/systemc/index.html

Event

Events are occurrence of signal values and changes.

- Events are meant to trigger processes of modules.

- An event has no duration or value.

It can be used for
- static sensitivity of processes, or
- dynamic sensitivity of processes

Event Trigger

Triggering events: event.notify()

- Events occur explicitly by calling .notify() method
- When an event notification is scheduled,
  the previous outstanding scheduled event is canceled

Canceling events: event.cancel()
Events can be explicitly canceled by calling .cancel() method

sc_event Queue

sc_event queue
- sc_event_queue lets a single event
  be scheduled repeatedly even for the same time
- when events are scheduled for the same time,
  each happens in a different delta cycle
- sc_event_queue objects do not support immediate notification
- .cancel() is replaced with .cancel_all()

Sensitivity

The **sensitivity** of a process instance is
the set of **events** and ***time-outs***
that can potentially cause the process to be resumed or triggered.

The **static sensitivity** of an unspawned process instance is
fixed *during elaboration*.

The **static sensitivity** of a spawned process instance is
fixed when the function *sc_spawn is called*.

The dynamic sensitivity of a process instance may
vary over time under the control of the process itself.

A process instance is said to be **sensitive to an event**
if the **event** has been added
to the static sensitivity or dynamic sensitivity of the process instance.

A **time-out** occurs when a given time interval has elapsed.

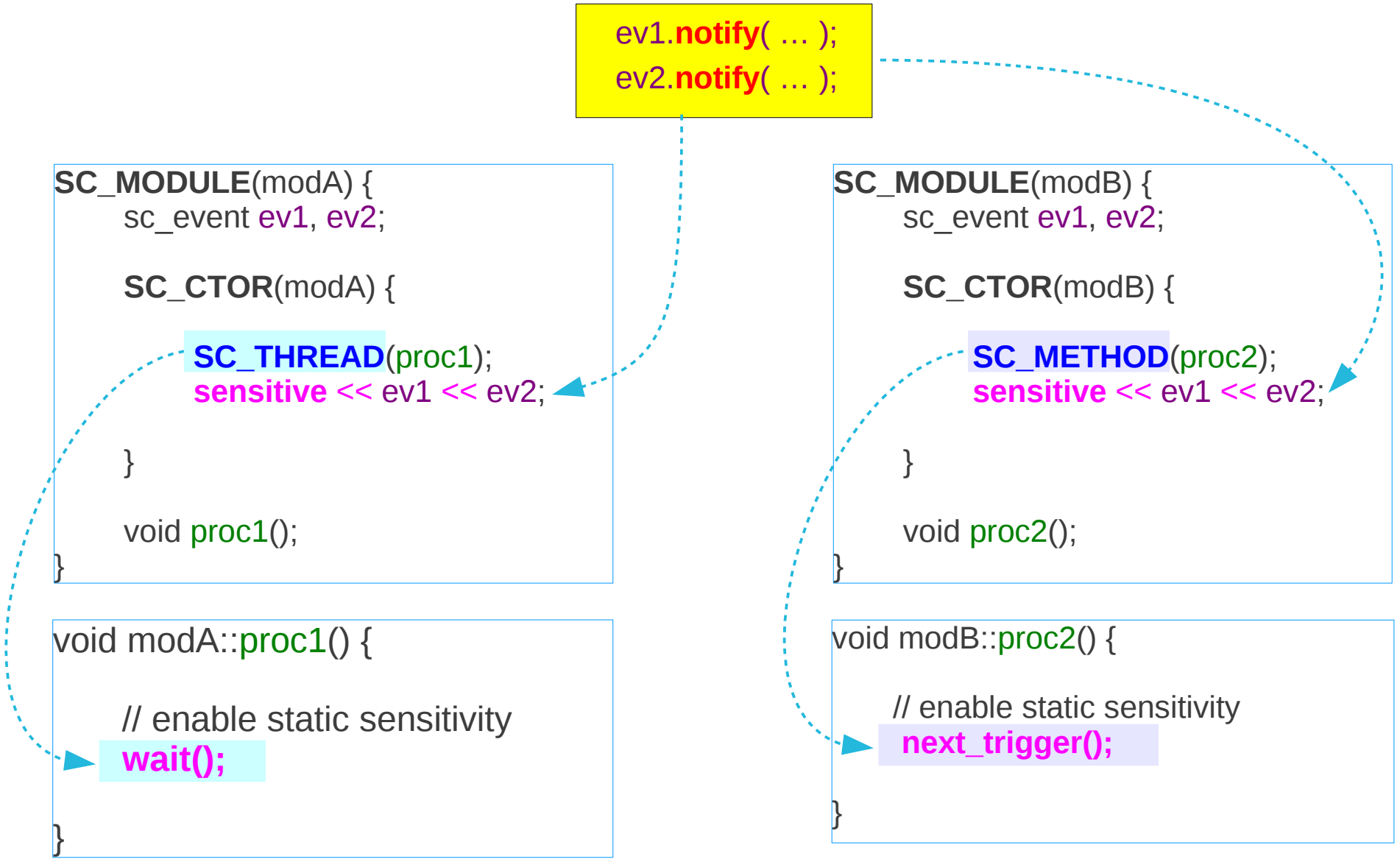
Static Sensitivity (1)

Data member **sensitive** of class sc_module can be used
to create the static sensitivity of an unspawned process instance
using operator<< of class sc_sensitive. (the only way)

However, static sensitivity may be **enabled** or **disabled**
By calling function **next_trigger**() (→ sc_method) or
By calling function **wait()** (→ sc_thread).

With no argument

Static Sensitivity (2)



Dynamic Sensitivity

```
ev1.notify( … );
ev2.notify( … );
```

```
SC_MODULE(modA) {
        sc_event ev1, ev2;

        SC_CTOR(modA) {

                SC_THREAD(proc1);
                sensitive << ev1 << ev2;
                // no need sensitive

        }

        void proc1();
}
```

```
void modA::proc1() {

        // enable static sensitivity
        wait(ev1 | ev2);

}
```

```
SC_MODULE(modB) {
        sc_event ev1, ev2;

        SC_CTOR(modB) {

                SC_METHOD(proc2);
                sensitive << ev1 << ev2;
                // no need sensitive
        }

        void proc2();
}
```

```
void modB::proc2() {

        // enable static sensitivity
        next_trigger(ev1 | ev2);

}
```

Dynamic Sensitivity – SC_THREAD (1)

SC_THREAD processes rely on **wait** method to **suspend** their execution

When the suspended process is reactivated,
it resumes execution at the statement after **wait**

Resumed when the kernel causes the process to **continue** execution,
starting with the statement immediately following the most recent call to **wait**.

When resumed, the process executes
until it reaches the next call to function **wait**.
Then, the process is **suspended** once again.

Dynamic Sensitivity – SC_THREAD (2)

wait(*time*);

wait(*event*);

wait(*event1* | *event2* | ...);

wait(*event1* & *event2* & ...);

wait(*timeout*, *event*);

wait(*timeout*, *event1* | *event2* | ...);

wait(*timeout*, *event1* & *event2* & ...);

Dynamic Sensitivity – SC_METHOD (1)

A method process instance
may have static sensitivity.

A method process, and only a method process,
May call the function **next_trigger** to create **dynamic sensitivity**.

next_trigger
does not suspend the process
temporarily sets a sensitivity list only for next time the process executes again
may be called repeatedly, overriding the previous calls

Without a next_trigger or a static sensitivity,
such process will never be executed again

Dynamic Sensitivity – SC_METHOD (2)

next_trigger(*time*);

next_trigger(*event*);

next_trigger(*event1* | *event2* | ...);

next_trigger(*event1 & event2 &* ...);

next_trigger(*timeout, event*);

next_trigger(*timeout, event1* | *event2* | ...);

next_trigger(*timeout, event1 & event2 &* ...);

References

[1] Aleksandar Milenkovic, 2002
CPE 626 The SystemC Language – VHDL, Verilog Designer's Guide
http://www.ece.uah.edu/~milenka/ce626-02S/lectures/cpe626-SystemC-L2.ppt

[2] Alexander de Graaf, EEMCS/ME/CAS, 2010
SystemC: an overview ET 4351
ens.ewi.tudelft.nl/Education/courses/et4351/SystemC-2010v1.pdf

[3] Joachim Gerlach, 2001
System-on-Chip Design with Systent of Computer Engineering
http://www2.cs.uni-paderborn.de/cs/ag-hardt/Forschung/Data/SystemC-Tutorial.pdf

[4] Martino Ruggiero, 2008
SystemC
polimage.polito.it/~lavagno/codes/SystemC_Lezione.pdf

[5] Deepak Kumar Tal, 1998-2012
SystemC Tutorial
http://www.asic-world.com/systemc/index.html